2nd Assignment report
“2-player Yavalath with Killer heuristic”

Miroslav Svítok

12250962

COMP30260

School of Computer Science and Informatics

College of Engineering, Mathematical & Physical Sciences

University College Dublin

25 November 2012

1 Introduction

Yavalath is an abstract board game for two players, designed by computer. It is the first game designed by computer using genetic programming. Rules are very simple and can be found at http://www.cameronius.com/games/yavalath/.

2 Implementation

I have implemented the game of Yavalath and experiments in Java programming language.

2.1 Structure of the program

The program is runnable, the game logic is provided in the main.java class. It interacts with user using command line. User always starts and can control the program by entering a coordinates of a move he wants to make.

2.1.1 Game board representation

The hexagonal game board and positions are represented by a single array of numbers. The size of the array is 121 elements, though only 61 cells correspond to playable cells. 60 are imaginary unplayable cells. These extra cells are needed because of the board printing and easy calculation of the adjacent cells (±1, ±10, ±11 are the indexes of adjacent cells in the array).

The array contains only numbers where 0 represents no playable cell, 1 represents empty cell and 2 and 3 represent players' cells.

2.1.2 User input

Input of a user is validated using regular expressions for correctness. If it's not valid, an exceptions is thrown. There are 3 types of exceptions:

BadMoveSyntaxException – in case of invalid input
NoPlayableMoveException – in case of non-existing cell (e.g. A6)
OccupiedCellException – when moving to an occupied cell

If the exception is thrown, program asks for the input again.

2.1.3 End of the game detection

The end of the game is detected checkForEnd() functions. It counts the number of user stones in every possible direction. In case of 3 cells in row the user loses, in case of 4 stones user wins.

2.1.4 Static evaluation function

I had decided to use finite automatons for searching for desired patterns in the game board. 3 automatons are used (this number can be probably reduced by joining them all together – but for the simplicity I created 3 separated). All of them are run twice for every direction and line in that direction. First time it is run for the active player and the second time for the opposite player. The evaluation function result is a difference between these two values.

I have evaluated patterns in following way:

- Empty-piece-piece-empty (-1)
- Empty-piece-empty-piece-empty (-1)
- Empty-piece-piece-empty-piece (+1)
- Empty-piece-emty-empty-piece (+1)

The winning positions are evaluated as +1000 and -1000 respectively according to a player who won/lost.

2.1.5 Game tree search and killer move heuristic

The game tree is searched via standard negamax search function with alpha/beta cutoff. The killer move heuristic enhancement is added too.

For every ply of a game tree a statistics of alpha-beta cut-offs is keeped. When a cut-off is done, the algorithm adds one point for the move which made the cut-off (on specific ply of a game tree). When getting a child of a node latter, the best move (killer move – the one that has the biggest amount of cut-offs) is placed on the first position of child nodes list. This step can increase chance of cut-off in the future.

3 Experiments

I performed systematic experimentation according to the assignment's instructions. I played three games of Yavalath for each depth limit (negamax search) with a computer. For the depth limit, I tested values of 3 and 4. Values of 5 and more are too time-consuming, because of the high branch factor of the game tree. There are 61 playable cells in Yavalath, so branching factor is 61 and less. If we assume the average branching factor to be 55, at the fifth level the game tree will have 55^5 = 503 284 375 nodes. This is a very high number (chess has the average branching factor about 35).

In following table are enlisted all the played games. First move is always Human (H) and second is computer (C). Column evaluations contains number of evaluations for both – search with and without killer move heuristic More detailed description is provided in the attached game_statistics.txt file.

Name (depth)	***Win***	***Evaluations (killer ON)***	***Evaluations (killer OFF)***	***Search disagree***	***Improvement***
Game A (d=3)	H	113298	129684	1	12.64 %
Game B (d=3)	H	153847	163783	3	6.07 %
Game C (d=3)	C	114523	161638	0	29.15 %
Game D (d=4)	C	2163104	4490713	1	51.83 %
Game E (d=4)	C	2000897	2298469	1	12.95 %
Game F (d=4)	C	2794585	5460307	6	48.82 %

When comparing a search with the killer move heuristic and without, it brings a significant improvement of the performance. The number of static evaluations performed is lower about from 6% to 50%. And the deeper search, the lower number of evaluations which means faster execution.

The algorithm with and without heuristic produced different results in some cases. The reason is there might be more best moves with the equal score – reordering can bring some of them before the move chosen by negamax without killer heuristic.

Increase of the search depth also brought better move selection – I wasn't able to beat computer in any game I played for d=4. One example of a better reasoning is that computer stopped putting his first move in the left upper corner (cell A1) – as it did all the time in games A, B, C.

4 Conclusion

The final decision about which algorithm to use is clear – negamax with killer move heuristic performed better.

Payoff for the better selection is much slower execution of the code. Even with killer move heuristic turned on, the algorithm was in case of d=4 quite slow (the algorithm run from 1 to 30 seconds). There is still need to improve the speed of the search with other enhancements to make it more useful.